

UNITED STATES PATENT APPLICATION

FOR

TECHNIQUES FOR STORING DATA ON
MESSAGE QUEUING MIDDLEWARE SERVERS
WITHOUT REGISTRATION OF THE SENDING APPLICATION

INVENTOR:

DAVID ARTHUR EATOUGH

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN, LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(503) 684-6200

EXPRESS MAIL NO. EL034437909US

TECHNIQUES FOR STORING DATA ON
MESSAGE QUEUING MIDDLEWARE SERVERS
WITHOUT REGISTRATION OF THE SENDING APPLICATION

FIELD OF THE INVENTION

[0001] The invention relates to network communications. More specifically, the invention relates to techniques for storing data on a middleware server.

BACKGROUND OF THE INVENTION

[0002] Store and forward message systems are commonly used for many applications, for example, electronic mail. Commercially available store and forward software applications include Message Queue Server from Microsoft Corporation of Redmond, Washington and the MQSeries of products available from International Business Machines of Armonk, New York.

[0003] These store and forward applications typically operate by an application that is sending a message establishing a connection with a server or other device that is to receive the message. These types of store and forward applications require substantial network, disk and memory resources. One reason that the network, disk and memory resources are consumed by store and forward applications is that a complete application program interface (API) is included in each application that provides store and forward functionality.

[0004] The store and forward application API is typically a full-function API that provides a specific method prescribed by a computer operating system or by another application program by which a programmer writing an application program can make requests of the operating system or another application. The full-function API allows an

application to provide store and forward functionality. However, in some situations a full-function store and forward API requires an unnecessarily large portion of available resources.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

Figure 1 is a conceptual illustration of one embodiment of a client and server coupled to provide a lightweight store and forward message queuing architecture.

Figure 2 is a block diagram of an electronic system.

Figure 3 is a flow diagram of one embodiment of lightweight store and forward message queuing.

DETAILED DESCRIPTION

[0005] Techniques for storing data on message queuing middleware servers without registration of the sending application are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention can be practiced without these specific details. In other instances, structures and devices are shown in block diagram form in order to avoid obscuring the invention.

[0006] Reference in the specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase “in one embodiment” in various places in the specification are not necessarily all referring to the same embodiment.

[0007] A lightweight store and forward architecture allows packets of data to be forwarded from an application running on a client system to a server system. An application that generates information in response to an event or other condition assembles a packet that includes a target identifier and data to be sent to the target. The packet is stored in a file in the client system. A messaging client forwards packets from the file to a messaging server on a target server system. The messaging server dispatches the information to a messaging handler to process the information. Thus, the application generating the packet is not required to handle network communications related to the packet and registration of a target application is not required.

[0012] When MSG enabled application 110 generates information that is to be sent to server 150, the information is encoded as a packet and the packet is stored in MSG file 120. The information stored in the packet can represent any type of electronic information. For example, execution errors or other events can generate a packet of information describing the nature and/or consequences of the event.

[0013] In one embodiment, MSG enabled application 110 acquires exclusive access to MSG file 120 for purposes of writing the packet to MSG file 120. After the packet is written to the MSG file 120, the exclusive access is relinquished. In one embodiment, the packet takes the following form:

Parameter	Format
Target ID	32-bit unsigned integer
Length	16-bit unsigned integer
Packet Data	Variable length binary data

Table 1: MSG packet format

[0014] In the embodiment of Table 1, the TargetID is a 32-bit value that represents the ultimate target of the packet. In alternate embodiments a different length value can be used as the TargetID. In the embodiment of Table 1, the Length is a 16-bit value representing the length, in bytes, of the data carried by the packet. Use of a 16-bit length value limits the packet data length to 65,535 bytes. In alternate embodiments a different value can be used to designate the packet data length if a different packet length limit is desired. Packet data represents the information carried by the packet.

[0015] MSG client 130 forwards packets stored in MSG file 120 to MSG server 160. In one embodiment, packets are forwarded using TCP/IP; however, any appropriate protocol known in the art can be used. Because MSG client 130 handles forwarding of packets, MSG enabled application 110 is required only to assemble packets and store the

packets in MSG file 120. MSG client 130 coordinates network communications. This frees MSG enabled application 110 to continue execution without being constrained by network communications.

[0016] Additional advantages of the client side architecture of Figure 1 is that the API included in MSG enabled application 110 is smaller and less complex than an API required for traditional store and forward messaging and the ultimate target of the packet is not required to be registered with the application that generates the target.

[0017] When a network connection is established between MSG client 130 and MSG server 160, MSG client 130 forwards the packet from MSG file 120. Upon receiving the packet, MSG server 160 dispatches the information stored in the packet to MSG registered handler 170. Multiple MSG handlers can exist to process different types of information or events that are stored in packets. In one embodiment, MSG handler 170 is a dynamic linked library (DLL) that is loaded based on a TargetID of a packet. The DLL then processes the packet data. Other techniques for processing the packet data can also be used.

[0018] When the packet data is dispatched to MSG handler 170, MSG server 160 sends an acknowledge (ACK) message to MSG client 130 identifying the packet that has been dispatched. Upon receipt of the ACK message MSG client 130 drops the packet from MSG file 120. In one embodiment, the dispatched packet is replaced by a null packet in MSG file 120 and, periodically, null packets are removed and the memory reclaimed. Dropping packets from MSG file 120 can be accomplished by other techniques as well.

[0019] **Figure 2** is a block diagram of an electronic system. The electronic system illustrated in Figure 2 is intended to represent a range of electronic systems, for example,

computer systems, set top boxes, PDAs. Alternative electronic systems can include more, fewer and/or different components.

[0020] Electronic system 200 includes bus 201 or other communication device to communicate information, and processor 202 coupled to bus 201 to process information. While electronic system 200 is illustrated with a single processor, electronic system 200 can include multiple processors and/or co-processors. Electronic system 200 further includes random access memory (RAM) or other dynamic storage device 204 (referred to as main memory), coupled to bus 201 to store information and instructions to be executed by processor 202. Main memory 204 also can be used to store temporary variables or other intermediate information during execution of instructions by processor 202.

[0021] Electronic system 200 also includes read only memory (ROM) and/or other static storage device 206 coupled to bus 201 to store static information and instructions for processor 202. Data storage device 207 is coupled to bus 201 to store information and instructions. Data storage device 207 such as a magnetic disk or optical disc and corresponding drive can be coupled to electronic system 200.

[0022] Electronic system 200 can also be coupled via bus 201 to display device 221, such as a cathode ray tube (CRT) or liquid crystal display (LCD), to display information to a computer user. Alphanumeric input device 222, including alphanumeric and other keys, is typically coupled to bus 201 to communicate information and command selections to processor 202. Another type of user input device is cursor control 223, such as a mouse, a trackball, or cursor direction keys to communicate direction information and command selections to processor 202 and to control cursor movement on display 221. Electronic

system 200 further includes network interface 230 to provide access to a network, such as a local area network.

[0023] Instructions are provided to memory from a storage device, such as magnetic disk, a read-only memory (ROM) integrated circuit, CD-ROM, DVD, via a remote connection (e.g., over a network via network interface 230) that is either wired or wireless, etc. In alternative embodiments, hard-wired circuitry can be used in place of or in combination with software instructions to implement the present invention. Thus, the present invention is not limited to any specific combination of hardware circuitry and software instructions.

[0024] A machine-accessible medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine (e.g., a computer). For example, a machine-accessible medium includes read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals); etc.

[0025] **Figure 3** is a flow diagram of one embodiment of lightweight store and forward message queuing. The flow of Figure 3 is described with respect to an event; however, messages can be generated in response to any type of condition. Events are detected at 310. In one embodiment, events are predetermined operating conditions; however, any type of condition can be detected at 310.

[0026] A packet is generated in response to the event at 320. In one embodiment, the packet is generated by an application that detects the event. In an alternate embodiment,

an application can detect the event and cause another application (e.g., an operating system) to generate the packet.

[0027] The packet is forwarded to a MSG file at 330. In one embodiment, the application generating the packet stores the packet in the MSG file and continues operation, if possible. Because the application only stores the packet in the MSG file on the same electronic device as the application is running, the API for message forwarding that is included in the application is much smaller than a full function store and forward API.

[0028] A messaging client application monitors the MSG file for addition of new packets and, at 340, forwards the packet(s) to a messaging server. The messaging client handles all network communications thus relieving the application of network communications related to the event that has been detected. The messaging server receives the packet from the messaging client via a network connection.

[0029] The messaging server dispatches the event to a messaging handler at 350. The messaging handler processes the information included in the packet that is related to the detected event to cause the server electronic system to take a predetermined action. In one embodiment the messaging handler is a DLL that is invoked in response to a predetermined TargetID carried by a packet. In alternate embodiments, the packet can be processed in a different manner.

[0030] At 360, the messaging server generates an acknowledge message in response to dispatching the packet to the messaging handler. The messaging client drops the packet from the MSG file, at 370, in response to the acknowledge message from the messaging server.

